1ª Parte

Suponha uma arquitectura sobre o modelo de 5 estados que consome programas constituídos por operações definidas por dois dígitos, exemplo de programa:

2 01 11 11 11 21 71 91

O primeiro valor, "2" é um valor que indica o instante em que o processo é inserido no estado NEW. Esse primeiro valor é descartado quando o programa é escrito em memória.

Escalonamento Round Robin, Quantum 4 mas configurável (#define). O estado ready só permite no máximo 4 processos (#define). Limitação esta aplicável apenas a processos vindos do estado NEW. Quando no mesmo instante, um processo vindo do NEW e do BLOCKED querem entrar no READY, o vindo do BLOCKED tem prioridade.

Nas restantes operações do programa, o primeiro dígito indica a instrução e o segundo indica a variável onde se aplica essa operação. Assim existem 10 operações diferentes e 10 variáveis disponíveis por programa.

O programa é guardado em memória, ficando o PCB apenas com a indicação da localização do programa e com o PC (Program Counter) do respectivo programa.

Tabela das instruções:

Códigos	Instruções	Significado
0X	ZERO X	X = 0
1X	ADD X	X ++
2X	SUB X	X --
3X	IF X	If X ==0, goto next line (PC++); else goto next line +1 (PC += 2)
4Y	BACK Y	Goto PC -= Y, em que Y é logo o valor do salto, não se vai consultar o valor da variável
5Y	FORW Y	Goto PC += Y, em que Y é logo o valor do salto, não se vai consultar o valor da variável
6X (facultativa)	FORK X	X = Fork() (facultativa)
7X	DISK SAVE X	Wait…
8X	COPY X	X0 = X
9X	EXIT	Exit

- Todas as instruções consomem um ciclo temporal
- As instruções de 0 a 5 e a 8 mantêm o processo em estado RUN
- A instrução (facultativa) 6 faz um fork duplicando o processo que irá para estado READY, esta função não é obrigatória de implementar, no entanto será um bónus para quem tiver tempo e se quiser aventurar ☺
- A instrução 7 simula uma escrita em disco envia o processo para estado BLOCK e consome 3 ciclos temporais em espera.
- A instrução 9 passa o processo para o estado EXIT

A memória onde as instruções são colocadas é estática e é representada por um array MEM[] que têm um limite de **300 posições**. (Este limite deve estar "#define" no vosso programa para permitir futuros ajustes).

Tenha em conta que qualquer programa em memória reserva sempre espaço para as 10 variáveis, mesmo que não as use todas.

O programa acima referido seria representado no array da memória MEM[] da seguinte forma:

	10 Variáveis reservadas					Instruções do Programa							
…	V0	V1	V2	…	V9	01	11	11	11	21	71	91	…

Exemplo de input:

1 01 11 11 11 21 71 91

2 01 12 13 14 24 71 90

3 01 11 11 31 41 71 91

Que obteria um print (no ciclo final) semelhante a:

V0 … V9 01 11 11 11 21 71 91 V0 … V9 01 12 13 14 24 71 90 V0 … V9 01 11 11 11 21 71 91

2ª Parte

É necessário que as instruções de cada processo sejam executadas sempre que esse processo se encontra em CPU.

A cada ciclo de um processo gasto em CPU provocará a execução de uma das suas instruções, fazendo avançar o Program Counter desse processo e actualizando as variáveis.

Assim, para os 10 tipos de instruções diferentes é necessário para cada um o respectivo método que executara as alterações das variáveis. (sendo a instrução fork optativa)

À medida que os processos terminam, o seu espaço em memória é libertado, fazendo com que a memória fique fragmentada.

Para lidar com isso, os novos processos a chegarem à memória tem de aproveitar esses espaços livres devido à memória ser limitada.

Para tal, pretende-se implementar dois gestores de memória:

Best Fit

Worst Fit

Deverá existir uma variável "Global" que define o qual gestor utilizar durante a execução.

Input

No Moodle encontra-se o ficheiro com o input de testes. Esse ficheiro contêm 8 linhas, sendo cada linha um processo.

A instrução fork apenas é evocada uma vez na última instrução, como é facultativa (mas com bónus), para quem a ignorar, remover do último processo a operação 61

Output

O trabalho deverá imprimir o conteúdo da memória de duas formas diferentes. Deverá existir uma variável "Global" que define qual o output.

Normal: Imprime a memória sempre após a inserção de um novo processo em memória e imprime sempre após a remoção de um processo da memória

Debug: Imprime a memória a cada ciclo, mas **apenas** nos primeiros 10 ciclos iniciais do vosso programa

Em ambos os modos anteriores, sempre que faz o print da memória, também deve imprimir o conteúdo dos 5 estados, para permitir ver por onde andam os processos.

Entrega

Duas datas de entrega sem qualquer penalização.

1ª data: entrega ate às 23h59 de 5 de Junho, discussão a 6 Junho

2ª data: entrega até às 23h59 de 29 de Junho, discussão a 30 Junho

A entrega será no Moodle e devera ser um .zip com os números de aluno no nome do ficheiro, ex "l23455_l33212_l4444.zip" e deverá conter o código fonte do trabalho assim como um relatório em PDF.

Trabalho pode ser individual ou em grupo até 3 pessoas no máximo.